Laborator si seminar Programare in Java si software matematic

Adrian Ichimescu

Adrianichimescu@gmail.com

Object-Oriented Programming

Object-oriented programming (**OOP**) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as *attributes* or *properties*), and code, in the form of procedures (often known as *methods*).

In OOP, computer programs are designed by making them out of objects that interact with one another.

OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

Significant object-oriented languages include: (list order based on TIOBE index) Java, Python, C++, C#

Object-Oriented Programming Benefits

OOP benefits:

Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.

Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.

Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.

Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace it, not the entire machine.

Object-Oriented Programming Concepts in Java

- Object
  An object is a software bundle of related state and behavior.
- Class
  A class is a blueprint or prototype from which objects are created.
- Inheritance
  classes inherit state and behavior from their superclasses
- Interface
  interface is a contract between a class and the outside world
- Package
  is a namespace for organizing classes and interfaces in a logical manner.

Classes and Objects

- **Object** – Objects have states (variables) and behaviors (methods)
- **Class** – A class can be defined as a template that describes the behavior/state that the object of its type support.
    - **Local variables** – Variables defined inside methods, constructors or blocks are called local variables.
    - **Instance variables** – Instance variables are variables within a class but outside any method.
    - **Class variables -** Class variables are variables declared within a class, outside any method, with the static keyword.

Exercise

Class Bicycle is a group of objects that have in common properties: states (fields: cadence, speed, gear) and behavior (methods: changeCadence,…) that define its interaction with the outside world

```
class Bicycle {
  int cadence = 0;
   int speed = 0;
   int gear = 1;
void changeCadence(int newValue) {
       cadence = newValue;   }
void changeGear(int newValue) {
       gear = newValue;   }
void speedUp(int increment) {
       speed = speed + increment;    }
    void applyBrakes(int decrement) {
       speed = speed - decrement;   }
    void printStates() {
       System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);   }
}
```

Exercise

```
class BicycleDemo {
  public static void main(String[] args) {
     // Create two different  Bicycle objects
    Bicycle bike1 = new Bicycle();
    Bicycle bike2 = new Bicycle();
     // Invoke methods on
    // those objects
    bike1.changeCadence(50);
    bike1.speedUp(10);
    bike1.changeGear(2);
    bike1.printStates();
     bike2.changeCadence(50);
    bike2.speedUp(10);
    bike2.changeGear(2);
    bike2.changeCadence(40);
    bike2.speedUp(10);
    bike2.changeGear(3);
    bike2.printStates();
  }
}
```

the **Bicycle** class does not contain a **main** method. That's because it's not a complete application; it's just the prototype for bicycles that might be used in an application. The responsibility of creating and using new Bicycle objects belongs to some other class in our application called **BicycleDemo**.

Bike1,2 are objects as an instance of Bicycle class.

Exercise

1) What is the output running BicycleDemo ?

2) Create new classes for a real-world Automobile object. Refer to the Bicycle class if you forget the required syntax.

Interface

- Class methods are the external way of interacting with class and its objects
- Interface commonly is the group of methods with empty bodies

```
interface Bicycle {
    void changeCadence(int newValue);
    void changeGear(int newValue);
    void speedUp(int increment);
    void applyBrakes(int decrement);
}
```

Interface

A class describes the attributes and behaviors of an object. An interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

An interface is different from a class in several ways, including –

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- A class can implement multiple interfaces.

Interface

If a class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.

```
class FSABicycle implements Bicycle {

 int cadence = 0;

 int speed = 0;

 int gear = 1;

 // The compiler will now require that methods

 // changeCadence, changeGear, speedUp, and applyBrakes

 // all be implemented. Compilation will fail if those

 // methods are missing from this class.

public void changeCadence(int newValue) {

    cadence = newValue;}

public void changeGear(int newValue) {

    gear = newValue;}

public void speedUp(int increment) {

    speed = speed + increment;}

public void applyBrakes(int decrement) {

    speed = speed - decrement;}

public void printStates() {

    System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);}

}
```

Exercise

- Write an application that create an object of FSABicycle class and prints its states

Exercise

- For the newly created **Automobile class**, create an **Interface** that defines its behavior, then require your class to implement it. Omit one or two methods and try compiling. What does the error look like?

Package

- A package is a namespace that organizes a set of related classes and interfaces.
- Java API is a collection of packages like String, Socket, File,…

https://docs.oracle.com/javase/8/docs/api/index.html

Exercise

- Play with importing previous package and call its class methods
- 3 importing options available:
  - The *Random* class is used to generate a stream of pseudorandom numbers, which is located in the *java.util* package. The *wildcard* character (*) is used to specify that all classes with that package are available to your program. This is the most common programming style.

    import java.util.; // Make all classes visible althow only one is used.*
  - Classes can be specified explicitly on *import* instead of using the wildcard character.

    import java.util.Random; // Make a single class visible.
  - We can the fully qualified class name without an import.

    java.util.Random.nextInt();

Inheritance

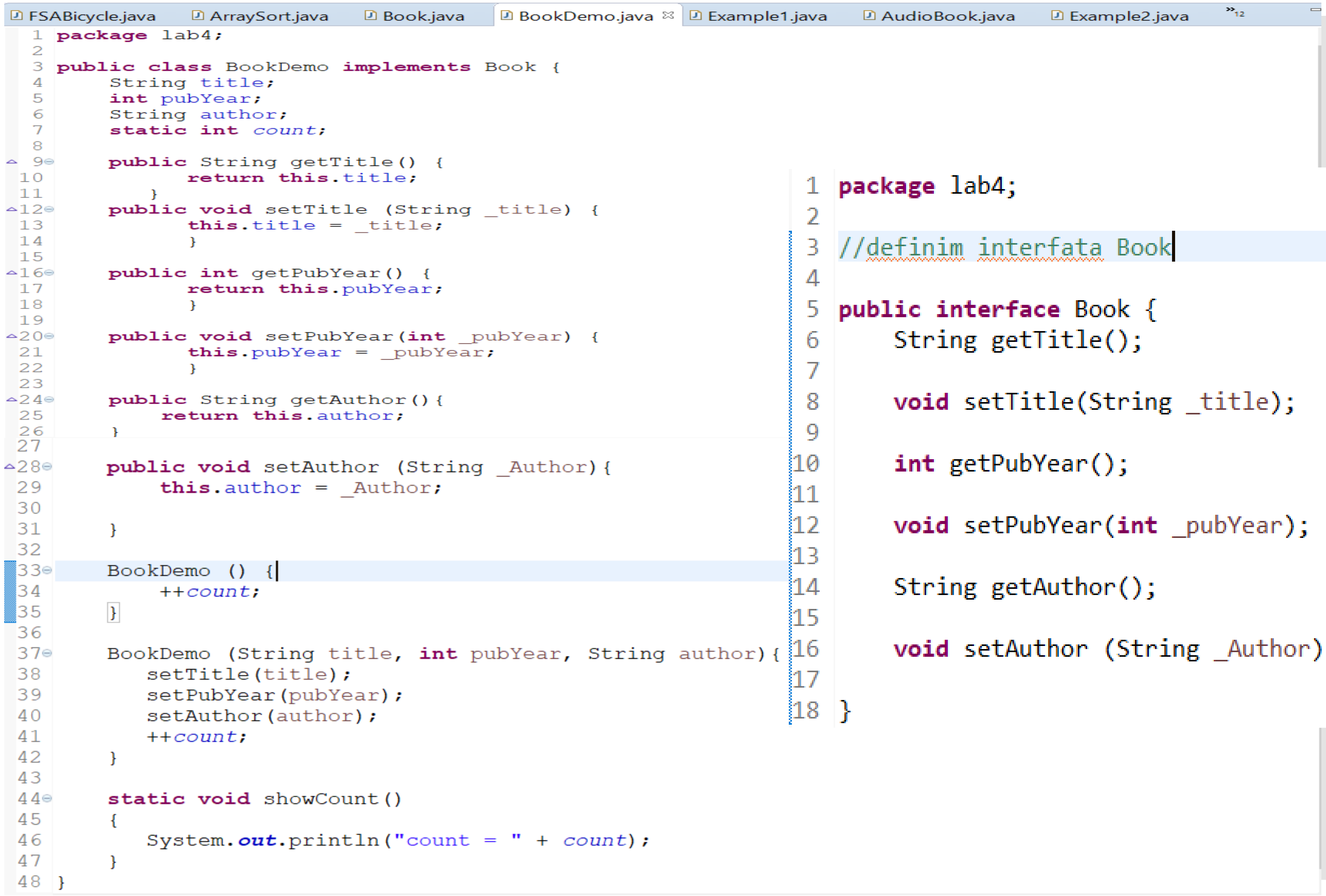
Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

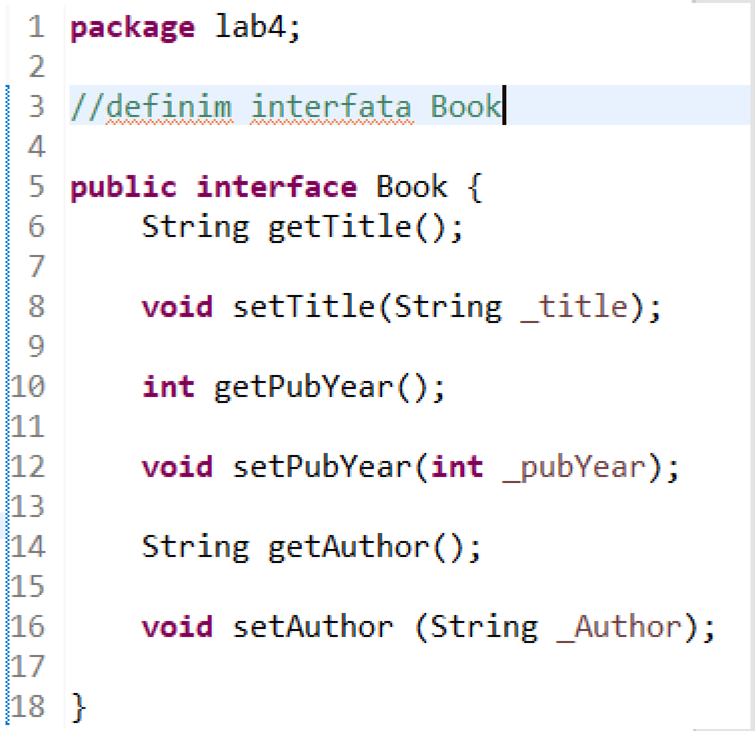
extends Keyword: **extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

```
class MountainBike extends Bicycle {

   // new fields and methods defining
   // a mountain bike would go here

}
```

Recap

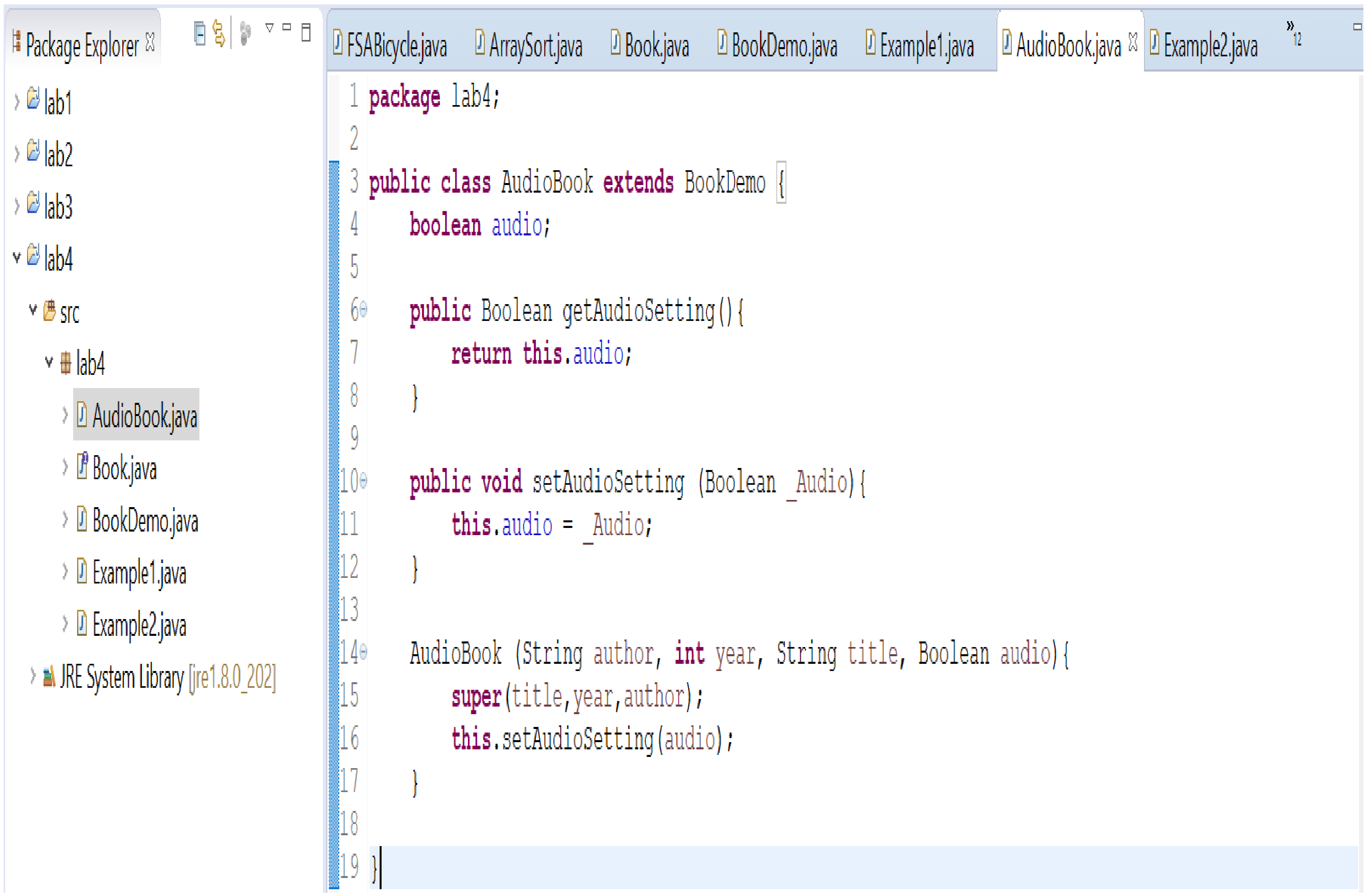


```java
package lab4;

public class BookDemo implements Book {
    String title;
    int pubYear;
    String author;
    static int count;

    public String getTitle() {
        return this.title;
    }
    public void setTitle (String _title) {
        this.title = _title;
        }

    public int getPubYear() {
        return this.pubYear;
        }

    public void setPubYear(int _pubYear) {
        this.pubYear = _pubYear;
        }

    public String getAuthor(){
        return this.author;
    }

    public void setAuthor (String _Author){
        this.author = _Author;

    }

    BookDemo () {
        ++count;
    }

    BookDemo (String title, int pubYear, String author){
        setTitle(title);
        setPubYear(pubYear);
        setAuthor(author);
        ++count;
    }

    static void showCount()
    {
        System.out.println("count = " + count);
    }
}
```



```java
package lab4;

//definim interfata Book

public interface Book {
    String getTitle();

    void setTitle(String _title);

    int getPubYear();

    void setPubYear(int _pubYear);

    String getAuthor();

    void setAuthor (String _Author);

}
```

Recap

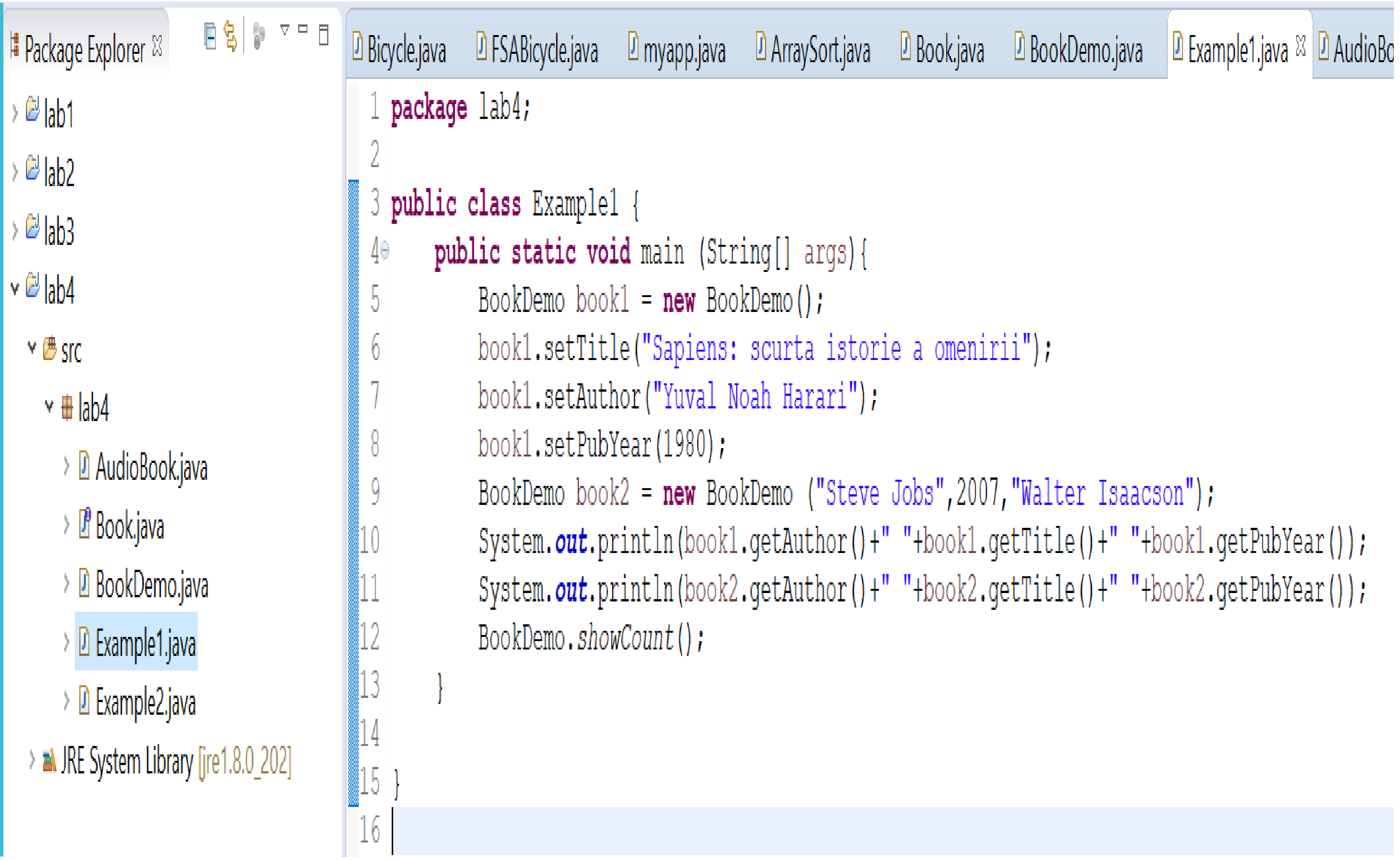


Package Explorer
- lab1
- lab2
- lab3
- lab4
  - src
    - lab4
      - AudioBook.java
      - Book.java
      - BookDemo.java
      - Example1.java
      - Example2.java
  - JRE System Library [jre1.8.0_202]

FSABicycle.java | ArraySort.java | Book.java | BookDemo.java | Example1.java | AudioBook.java | Example2.java

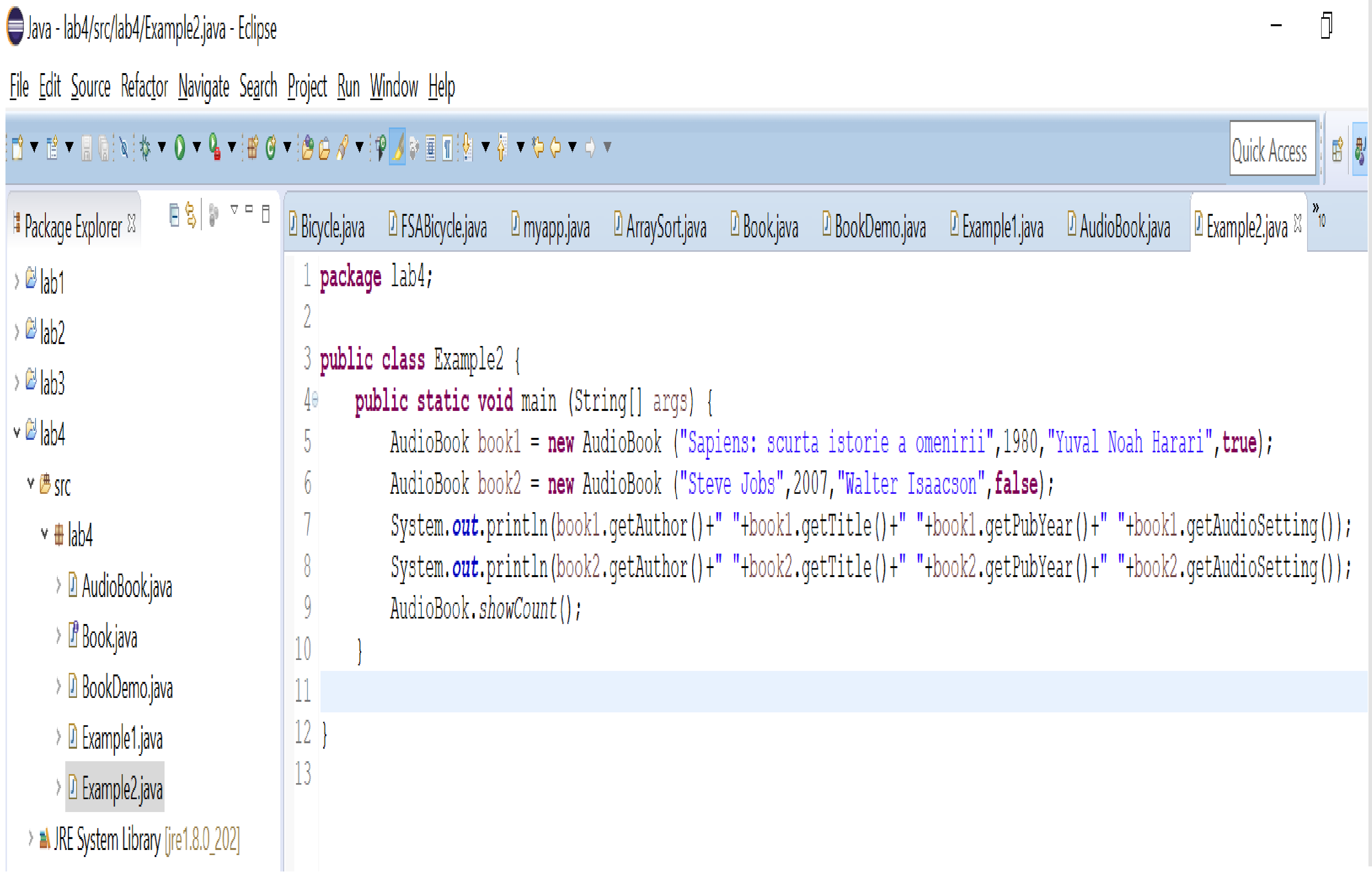
```java
package lab4;

public class AudioBook extends BookDemo {
    boolean audio;

    public Boolean getAudioSetting(){
        return this.audio;
    }

    public void setAudioSetting (Boolean _Audio){
        this.audio = _Audio;
    }

    AudioBook (String author, int year, String title, Boolean audio){
        super(title,year,author);
        this.setAudioSetting(audio);
    }

}
```

Recap



```java
package lab4;

public class Example1 {
    public static void main (String[] args){
        BookDemo book1 = new BookDemo();
        book1.setTitle("Sapiens: scurta istorie a omenirii");
        book1.setAuthor("Yuval Noah Harari");
        book1.setPubYear(1980);
        BookDemo book2 = new BookDemo ("Steve Jobs",2007,"Walter Isaacson");
        System.out.println(book1.getAuthor()+" "+book1.getTitle()+" "+book1.getPubYear());
        System.out.println(book2.getAuthor()+" "+book2.getTitle()+" "+book2.getPubYear());
        BookDemo.showCount();
    }

}
```

Recap



```java
package lab4;

public class Example2 {
    public static void main (String[] args) {
        AudioBook book1 = new AudioBook ("Sapiens: scurta istorie a omenirii",1980,"Yuval Noah Harari",true);
        AudioBook book2 = new AudioBook ("Steve Jobs",2007,"Walter Isaacson",false);
        System.out.println(book1.getAuthor()+" "+book1.getTitle()+" "+book1.getPubYear()+" "+book1.getAudioSetting());
        System.out.println(book2.getAuthor()+" "+book2.getTitle()+" "+book2.getPubYear()+" "+book2.getAudioSetting());
        AudioBook.showCount();
    }

}
```